

Software & Data Integrity Failures

Software & Data Integrity Failures

- Code and infrastructure not protected against integrity violations, e.g.
 - application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs)
 - insecure CI/CD pipeline introduces the potential for unauthorized access, malicious code, or system compromise
 - auto-update functionality downloading and applying updates without sufficient integrity verification
- Objects or data encoded or serialized into a structure that an attacker can see and modify is vulnerable to insecure deserialization, e.g. insecure native serialization formats and libraries being used

Data Factors

A08:2021 – Software and Data Integrity Failures

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Max Coverage	Avg Coverage	Total Occurrences	Total CVEs
10	16.67%	2.05%	6.94	7.94	75.04%	45.35%	47,972	1,152

Prevention

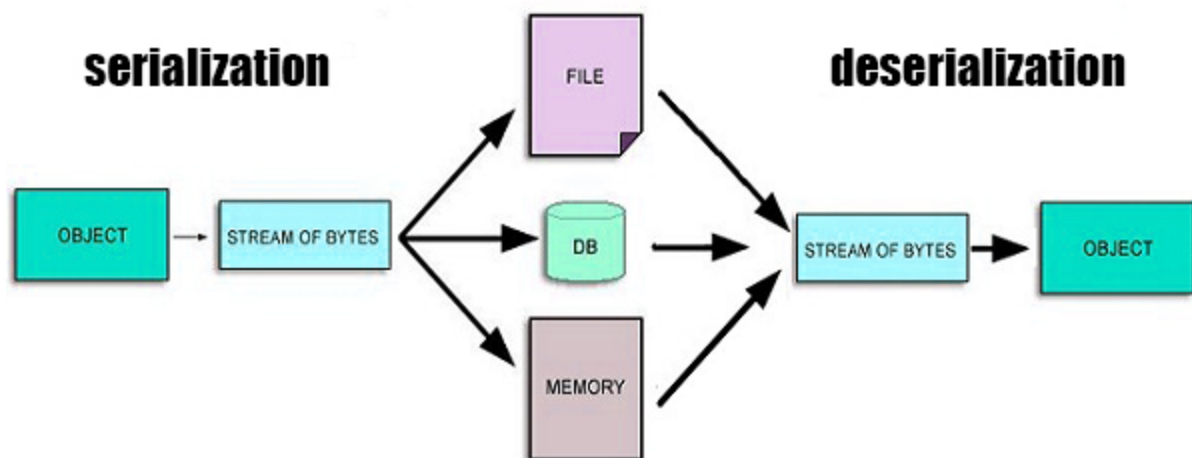
- **Using digital signatures** to verify the software or data is unaltered and from the expected source
- Ensuring libraries and dependencies (e.g. `npm` or Maven) **consume trusted repositories**
- Hosting an internal known-good and vetted repository as a proxy
- Using a software supply chain security tool (e.g. [OWASP Dependency Check](#) or [OWASP CycloneDX](#)) to verify that components do not contain known vulnerabilities

- Establishing a review process for code and configuration changes to minimize the chance that malicious code or configuration could be introduced into the software pipeline
- Establishing a CI/CD pipeline with proper segregation, configuration, and access control to ensure the integrity of the code flowing through the build and deploy processes

Deserialization

Serialization

Object serialization transforms an object's data to a bytestream that represents the state of the data. The serialized form of the data contains enough information to recreate the object with its data in a similar state to what it was when saved. [¹]



Deserialization

```
InputStream is = request.getInputStream();  
ObjectInputStream ois = new ObjectInputStream(is);  
AcmeObject acme = (AcmeObject)ois.readObject();
```

- The casting operation to `AcmeObject` occurs **after** the deserialization process ends
- It is not useful in preventing any attacks that happen during deserialization from occurring

Insecure Deserialization

- Insecure deserialization often leads to **remote code execution (RCE)**, one of the most serious attacks possible
- Other possible attacks include
 - replay attacks
 - injection attacks
 - privilege escalation
 - DoS

Exercise 8.2

1. What happens when the `root` object would be deserialized?

```
ArrayList<Object> root = new ArrayList<>(Integer.MAX_VALUE);
```

Exercise 8.3

1. What happens when the `root` object would be deserialized?

```
Set root = new HashSet();
Set s1 = root;
Set s2 = new HashSet();
for (int i = 0; i < 100; i++) {
    Set t1 = new HashSet();
    Set t2 = new HashSet();
    t1.add("foo");
    s1.add(t1);
    s1.add(t2);
    s2.add(t1);
    s2.add(t2);
    s1 = t1;
    s2 = t2;
}
```

Prevention

- **Avoid native deserialization formats 100**
 - JSON/XML lessens (but not removes) the chance of custom deserialization logic being maliciously repurposed
- Use the Data Transfer Object (DTO) pattern
 - Exclusive purpose is data transfer between application layers

If serialization cannot be avoided

- Sign any serialized objects & only deserialize signed data
- Enforce strict type constraints during deserialization before object creation (Not sufficient on its own!)
- Isolate deserialization in low privilege environments
- Log deserialization exceptions and failures
- Restrict or monitor incoming and outgoing network connectivity from containers or servers that deserialize
- Monitor & alert if a user deserializes constantly

✓ SerialKiller (Java)

Replacing every `java.io.ObjectInputStream` instantiation

```
ObjectInputStream ois = new ObjectInputStream(is);  
String msg = (String) ois.readObject();
```

with `SerialKiller` from a look-ahead Java deserialization library

```
ObjectInputStream ois = new SerialKiller(is, "/etc/serialkiller.conf");  
String msg = (String) ois.readObject();
```

secures the application from untrusted input. Via `serialkiller.conf` classes can be block- or allowlisted.

✗ node-serialize (JavaScript)

The `node-serialize` module uses `eval()` internally for deserialization, allowing exploits like












```
var serialize = require('node-serialize');  
var x = '{"rce": "_$$ND_FUNC$$_function (){console.log(\'exploited\')}{()}" }'  
serialize.unserialize(x);
```


⚠ *The affected version `0.0.4` of `node-serialize` is also the latest version of this module!*

Exercise 8.4 (🏠)

1. Report at least one of two [typosquatting](#) dependencies that the Juice Shop fell for (★★★★ - ★★★★★★)
2. Report another vulnerability that could be exploited in a [Software Supply Chain Attack](#) (★★★★★)

Exercise 8.5 ()

1. Find the „NextGen“ successor to the half-heartedly deprecated XML-based B2B API
 - This new API uses a popular standard for REST API specification & documentation
2. Exploit this API with at least one successful DoS-like Remote Code Execution ( 
   -      )

 *If the server would need >2sec to process your attack request, it is considered „DoS-like“ enough.*